Computer Science Department

TECHNICAL REPORT

FINDING EULER TOURS IN PARALLEL[a]

By

Mikhail Atallah

and

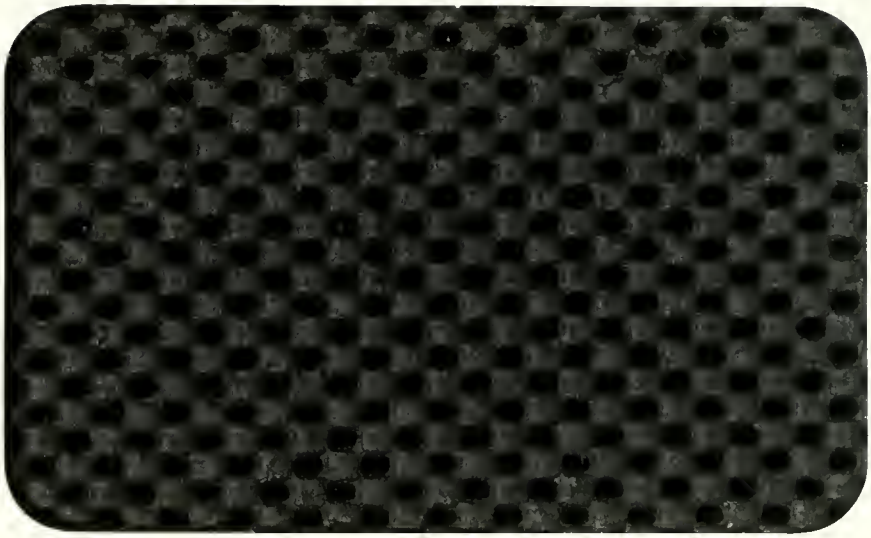
Uzi Vishkin

Technical Report #134
Ultranote #73

NYU/CompSci/TR-134 c.1
Atallah
Finding Euler tours in
parallel.

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



FINDING EULER TOURS IN PARALLEL[a]

By

Mikhail Atallah

and

Uzi Vishkin

Technical Report #134
Ultranote #73

Copy 1

Finding Euler Tours in Parallel[a]

November 1982. First revision: February 1983. Second revision: May 1984.

Mikhail Atallah

Department of Computer Science

Purdue University

West Lafayette, Indiana 47907

and

Uzi Vishkin[b]

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

251 Mercer St., New York, NY 10012

Abstract.

The problem of finding Euler tours in directed and undirected Euler graphs is considered. We give O(log |V|) time algorithms using a linear number of processors on a concurrent-read concurrent-write parallel RAM.

(a) A first version of this research appeared in [Atallah].

(b) This research was supported by DOE grant DE-AC02-76ER03077 and by NSF grant NSF-MCS79-21258.

I. Introduction

The family of models of computation used in this paper is the parallel random-access-machines (PRAMs). All members of this family employ p synchronous processors all having access to a common memory. We mention three members of the PRAM family in descending order of strength. In a concurrent-read concurrent-write (CRCW) PRAM, simultaneous reading from the same memory location is allowed as well as simultaneous writing. In the latter case the lowest numbered processor succeeds. A concurrent-read exclusive-write (CREW) PRAM allows simultaneous reading into the same memory location but not simultaneous writing. An EREW PRAM does not allow simultaneous reading or writing. See [Vishkin83b] for a recent survey of results concerning the PRAM family.

In the present paper we consider the problem of finding Euler tours in directed and undirected Euler graphs. Parallel algorithms which run in time O(log |V|) using a linear number of processors on a CRCW PRAM are given.

Consider an Euler digraph obtained from a tree by replacing each one of its edges by two anti-parallel edges. It was known how to compute an Euler circuit of such a graph. The main idea of the algorithm is quite elegant: We find a spanning tree of some auxiliary graph and an Euler circuit of the Euler digraph obtained from the tree. Then we expand this Euler circuit to an Euler tour of the graph.

II. Basics

The following definitions and propositions follow [Even]. Let H(U,F) be a finite undirected graph. An <u>Euler path</u> of H is a path $e_1, e_2, \ldots, e_{|F|}$ such that every edge appears on it exactly once. An undirected graph which has an Euler path is an <u>Euler graph</u>.

<u>Proposition 1</u>. H is an Euler graph iff it is connected and exactly two vertices are of odd degree or all vertices are of even degree. In the latter case, every

Euler path of the graph is a circuit and in the former case every Euler path leads from one of the two odd degree vertices to the other.

Let $G(V,E)$ be a finite directed graph. An <u>Euler path</u> of G is a path $e_1, e_2, \ldots, e_{|E|}$ such that every edge appears on it exactly once. A directed graph which has an Euler path is an <u>Euler digraph</u>. The <u>underlying undirected graph</u> of a digraph is the graph resulting from the digraph if the direction of the edges is ignored. We use $\deg_{out}(v)$ (resp. $\deg_{in}(v)$) to denote the in-degree (resp. out-degree) of vertex v.

<u>Proposition 2</u>. G is an Euler digraph iff its underlying graph is connected and one of the two following conditions holds:

(1) There are two vertices u and v such that, $\deg_{out}(u) = \deg_{in}(u) + 1$ and $\deg_{out}(v) + 1 = \deg_{in}(v)$, while for all other vertices w, $\deg_{out}(w) = \deg_{in}(w)$. In this case every Euler path starts at u and ends at v.

(2) For all vertices w, $\deg_{out}(w) = \deg_{in}(w)$. In this case every Euler path is a circuit.

For both directed and undirected graphs we restrict, w.ℓ.g., the discussion to inputs which are Euler graphs in which every Euler path is a circuit. Let us justify it: (1) Given an input graph we can check within the efficiency bound claimed in this paper if it is an Euler graph, if it has Euler circuits and, if it has Euler paths but not Euler circuits, what are the endpoints of these paths. For this apply the above propositions. The only non-trivial detail is checking connectivity of underlying undirected. For this use the $O(\log n)$ time algorithm of [Shiloach and Vishkin] which uses a linear number of processors. (2) If we are given an Euler graph (or digraph) in which every Euler path is not a circuit, it is possible to "add" a single edge to it such that in the new graph every Euler path is a circuit. After finding an Euler circuit "delete" the added edge to obtain an Euler path in the original graph.

Note. Whenever specifying the number of processors used we ignore the constant factor. Namely, instead of "O(t) time using O(p) processors" we write "O(t) time using p processors". Since we can always save a constant factor in the number of processors at a cost of the same constant factor in the running time.

Subsection III.1 gives a high-level description of the algorithm for finding Euler circuits in digraphs. A detailed description follows in Subsection III.2. Section IV shows how to extend the algorithm of Section III for finding Euler circuits of undirected graphs.

III. Euler Digraphs

III.1 A high-level description of the algorithm

Step 1. Partition the edges of the input graph $G=(V,E)$ into pairwise disjoint circuits.

Step 2. Compute an auxiliary undirected bipartite graph $G_1=(V_1,E_1)$ defined as follows. G_1 has two sets of vertices: "circuit-vertices" and "real-vertices". Each circuit of Step 1 is a circuit-vertex of G_1. Each vertex of G is a real-vertex of G_1. There is an edge between a real-vertex and a circuit-vertex if the (corresponding) vertex lies on the (corresponding) circuit in G.

Step 3. Find a spanning tree $T=(V_1,E_1')$ of G_1. Replace each edge of T by two anti-parallel edges (with the same end-points) to obtain an Euler digraph $\tilde{T}$.

Step 4. Find an Euler circuit of $\tilde{T}$ and use it to guide the stitching of the circuits that were found in Step 1 into an Euler circuit of G.

Before proceeding to a detailed description of the algorithm we summarize a solution of [Tarjan and Vishkin] for computing an Euler circuit of $\tilde{T}$. The circuit is computed into a vector FOLLOW. For each vertex v of V_1 we do the following. Let deg(v) = d in T and let the d adjacent edges of v in T be $\{v,u_0\},\ldots,\{v,u_{d-1}\}$.

Assign FOLLOW$((u_i,v)) := (v,u_{i+1 \bmod d})$ for $0 \le i \le d-1$.

(This is called a <u>circular-succession assignment</u>)

FOLLOW gives for each edge of $\tilde{T}$ a successor. Thus, given an edge of $\tilde{T}$ FOLLOW gives a path that starts at this edge. It is easy to verify that FOLLOW gives an Euler circuit of $\tilde{T}$. Step 4.1 in the lower-level description that follows is of particular interest. There, we describe how the circular succession assignment is actually obtained.

III.2 The algorithm

<u>Input form</u>. The vertices of G are represented by the numbers $1,\ldots,n$. The edges are represented by a vector of length $|E|=m$.

We employ $n + m$ processors in the CRCW PRAM.

<u>Step 1.1</u>. Sort all edges (i,j) in E by the lexicographic order (defined as, $(a,b) < (c,d)$ if $a < b$ or $a = b$ and $c < d$). Let OUT-EDGE(v,k) be the k-th outgoing edge of v, $1 \le k \le \deg_{out}(v)$. Sort all edges (i,j) in E in the reversed lexicographic order (defined as the lexicographic order on (j,i)). Let IN-EDGE(v,k) be the k-th incoming edge to v, $1 \le k \le \deg_{in}(v)$.

<u>Implementation Remark 1</u>. We can apply the sorting algorithm of [Ajtai, Komlos and Szemeredi] for these sorts. This algorithm runs in $O(\log m)$ time using m processors and $O(m)$ space on an EREW PRAM. The constants in front of log m are large. It is possible to get $O(\log m)$ time (with moderate constants) using m processors but $O(nm)$ space using "orthogonal trees" (see [Thompson] for more on this simple algorithm). We avoid elaborating on this. A third possiblity is to apply the randomized sorting algorithm of [Reif and Valiant]. It runs in time $O(\log m)$ almost surely, using m processors.

<u>Step 1.2</u>. <u>for all</u> v and $1 \le k \le \deg_{in}(v)$ <u>pardo</u>

SUCC(IN-EDGE(v,k)) := OUT-EDGE(v,k); NEXT(IN-EDGE(v,k)) := OUT-EDGE(v,k)

Explanation. Step 1.2 completes partitioning the edges of G into circuit as required by Step 1 above. The vector SUCC (and the auxiliary vector NEXT) defines a successor for each edge in E such that each edge in E has exactly one predecessor. Therefore the paths defined by SUCC partition the edges of E into (disjoint) circuits. Our goal in the following steps is to correct SUCC to contain an Euler circuit. At this stage the auxiliary vector NEXT is an exact copy of SUCC. The reason for using NEXT is that for later in the computation we need the present values of SUCC, while the computation of Step 2.1 modifies NEXT.

Complexity. O(1) time, m processors.

Step 2.1. Select the edge of minimum lexicographic order of each circuit to "represent" it in the auxiliary graph G_1. For this we use the vector D. D((i,j)) is initialized to (i,j), for all (i,j) ϵ E. The iterations below use the "doubling technique" ([Wyllie]).

Apply $\lceil \log m \rceil$ iterations:

for every edge e pardo

D(e) := MIN{D(e),D(NEXT(e))}; NEXT(e) := NEXT(NEXT(e)).

D((i,j)) of all edges of the same circuit have now the edge of minimum lexicographic order of the circuit. ($\lceil \log m \rceil$ iterations sufficed since none of the circuits has more than m edges).

An edge e represents the circuit it lies on if e=D(e).

Complexity. O(log m) time, m processors.

Step 2.2. Compute the auxiliary bipartite undirected graph $G_1(V_1,E_1)$.

V_1 = {v; v is a vertex in V or it is a circuit defined by SUCC}

Recall that the first kind of vertices are called real-vertices while the second

are called circuit-vertices. (Note. In our writing we use the following conventions. We do not always distingiush between real-vertices of G_1 and vertices of G. We also do not distinguish among a circuit that was found in Step 1.2, the corresponding circuit-vertex of G_1 and the edge representing this circuit in the computation.)

There is an edge between a circuit-vertex and each of the real-vertices on it:

$E_1 = \{\{v,w\}$; w is a circuit-vertex, v is a real-vertex, and v lies on the circuit w in $G\}$

Let $\{v,w\}$ be an edge in E_1 where v is a real-vertex and w is a circuit-vertex. Observe that the vertex v may appear on the circuit w more than once. At each appearance v is entered through another edge of the form (i,v) (in G). Let CERTIFICATE($\{v,w\}$) be one of the edges of the form (i,v) on circuit w. (The reason for the name is that it certifies the edge $\{v,w\}$).

Implementation. A proper application of sorting (along similar lines to Implementation Remark 1) would readily yield the graph G_1 and selection of edges for CERTIFICATE($\{v,w\}$), for all $\{v,w\} \in E_1$. Complexity. The graph G_1 has $O(n+m)$ vertices and $O(n+m)$ edges. Hence Step 2.1 takes $O(\log n)$ time using $n+m$ processors.

Step 3.1. A spanning tree $T = (V_1, E_1')$ of G_1 is computed. For this we use an adaptation of the connectivity algorithm of [Shiloach and Vishkin] as described in [Tarjan and Vishkin]. Complexity. $O(\log n)$ time using $n+m$ processors on a CRCW PRAM.

Step 3.2. Obtain the digraph $\tilde{T}$ from T by replacing each edge $\{v,w\}$ by a pair of anti-parallel edges (v,w) and (w,v). $\tilde{T}$ is an Euler digraph since $\deg_{in}(v) = \deg_{out}(v)$ for every vertex v in $\tilde{T}$.

Complexity. Implementation remark 1 bounds Step 3.2 to $O(\log n)$ time and n processors.

The goal of Step 4.1 is to compute into the vector SUCC a large circuit which will alternate between edges of G and edges of $\tilde{T}$ and have two properties: (1) The edges of $\tilde{T}$ will appear on it in an order of an Euler circuit of $\tilde{T}$. (2) The edges of G will appear on it in an order of an Euler circuit of G. We concentrate on achieving the first property. Interestingly, the second property will follow without further care.

Step 4.1. Recall the idea of finding an Euler circuit of $\tilde{T}$ using circular succession assignments.

It is trivial to obtain circular succession for each real-vertex v of V_1: Let deg(v) = d in T and let the d adjacent edges of v in T be $\{v,u_0\},\ldots,\{v,u_{d-1}\}$. Set $SUCC((u_i,v)) := (v,u_{i+1 \bmod d})$, for $0 \le i \le d-1$.

It is more interesting to obtain circular succession for each circuit-vertex w of V_1.

Let deg(w) = d in T and let the d adjacent edges of w in T be $\{v_0,w\},\ldots,\{v_{d-1},w\}$. Recall that $v_0,\ldots,v_{d-1}$ are real-vertices. Let (i_α,v_α) be CERTIFICATE($\{v_\alpha,w\}$), for $0 \le \alpha \le d-1$. All these d certificate edges appear on the circuit w (in G) in some circular order. The key idea in this step is to use this circular order in order to obtain a circular succession with respect to $\tilde{T}$. This is done as follows (Say that following Step 1 $SUCC((i_\alpha,v_\alpha)) = (v_\alpha,j_\alpha)$, $0 \le \alpha \le d-1$. That is, (v_α,j_α) is the successor of (i_α,v_α) in the circuit w):

$$SUCC((v_\alpha,w)) := (v_\alpha,j_\alpha);\ SUCC((i_\alpha,v_\alpha)) := (w,v_\alpha),\ \text{for } 0 \le \alpha \le d-1.$$

We just set the successor of (v_α,w) to be (v_α,j_α). We proceed along the circuit w (on the path defined by SUCC) till we hit an edge (i_β,v_β) which is a certificate

edge for some edge $\{w,v_\beta\}$ in T. (It might be (v_α, i_α) itself). SUCC will lead us from (i_β, v_β) to (w,v_β). If we ignore the edges of w then (w,v_β) is the successor of (v_α,w). Thus, we obtained circular succession relative to circuit-vertices. Therefore, SUCC contains an Euler circuit with respect to $\tilde{T}$ as required (in property (1) above).

Claim. SUCC contains an Euler circuit with respect to G.

Proof. Let (i,j) be an edge of G. we show that it is on the circuit defined by SUCC. Following Step 1 (i,j) is on some circuit w. Say that w is a circuit-vertex of degree d in T. The circuit w contains d certificate edges which partition the edges of w into d paths: Each such path starts at the successor of a certificate edge and ends at the next cetificate edge on w. (i,j) must lie on one such path. Each cetificate edge must lie on the circuit defined by SUCC. Hence, (i,j) must lie on the circuit defined by SUCC. Also, a second appearance of this path must follow the traversal of all other edges of G or otherwise we get a contradiction to the circuit of SUCC being an Euler circuit with respect to $\tilde{T}$. ∘

complexity. O(1) time, n processors.

Finally, we need to "clean" the circuit of SUCC from the edges of $\tilde{T}$. Observe that there are no more than two successive edges of $\tilde{T}$ in the circuit defined by SUCC. Therefore, Step 4.2 will finish the job.

Step 4.2. For all edges e of G perform two "doubling" iterations:

if SUCC(e) is an edge of $\tilde{T}$

then SUCC(e) := SUCC(SUCC(e))

Complexity. O(1) time, m processors.

Complexity of the whole algorithm. O(log n) using n+m processors. Subject to Implementation Remark 1 we can limit the space to O(n+m) as well.

IV. Undirected Graphs

Let H be an Euler undirected graph. We show below how to orient the edges of H in a way which result in an Euler digraph. This solves our problem since now we can apply the algorithm of the previous section to this digraph.

Let H' be a directed graph obtained from H by replacing each edge $\{u,v\}$ of H by two anti-parallel edges (u,v) and (v,u). Our problem is to decide which one of these directed edges will replace $\{u,v\}$. Let v be a vertex of H. Say that its degree is d where d is some even number. Let $\{v,u_1\},\{v,u_2\},\ldots,\{v,u_d\}$ be the edges adjacent to v. We partition the edges of H' into circuits as follows.
For every odd number i, $1 < i < d$, assign

$$\mathrm{NEXT}((u_i,v)) := (v,u_{i+1});\ \mathrm{NEXT}((u_{i+1},v)) := (v,u_i)$$

Observe that the circuits defined by the vector NEXT are partitioned into pairs as follows: for every circuit C there is another circuit C' which contains exactly all the edges which are anti-parallel to those of C. Our goal is to delete all edges of C or C' from H' for every pair of circuits. For each $e \in H'$ compute into D(e) the minimum edge (i,j) in the lexicographic order in its circuit. (Apply a computation similar to Step 2.1). Now for each edge (u,v) in H, if $D((u,v)) < D((v,u))$ then (u,v) is selected and (v,u) is deleted. For each pair of circuits this results in selecting all edges of one and deleting all edges of the other. We indeed get an Euler digraph since the fact that H satisfied Proposition 1 implies that the digraph satisfies Proposition 2. This computation takes O(log n) time using m processors and so does the application of the algorithm of the previous section to the Euler digraph. Again, O(n+m) space suffices.

Conclusion

Using general simulation results of [Eckstein] or [Vishkin83a] it is possible

to run our algorithms on an EREW PRAM (or the stronger CREW PRAM) in time $O(\log^2 n)$ using a linear number of processors.

The only place where our algorithms used simultaneous access to the memory location is in employing the Shiloach-Vishkin connectivity algorithm (Step 3.1).

Recently, [Awerbuch, Israeli and Shiloach] obtained independently results which are similar to the present paper. We feel that our solutions for both the directed and undirected case are simpler than theirs. Theirs and ours algorithms for the directed case are pretty similar. However, our presentation is slightly simplified due to the observation that it is possible to use the known solution of [Tarjan and Vishkin] for finding an Euler circuit in a tree where each edge is replaced by two anti-parallel edges. We use it both for the intuitive explanation of the algorithm and as its main step. In the undirected case they claim that it is not that easy to find 'Eulerian' orientation in logarithmic time. However, our solution for this problem is very short and simple.

REFERENCES

[Atallah] M.J. Atallah, "Finding Euler tours in parallel", Proc. Seventeenth Annual Conf. on Information Sciences and Systems, 1983, 685-689.

[Awerbuch, Israeli and Shiloach] B. Awerbuch, A. Israeli and Y. Shiloach, "Finding Euler circuits in logarithmic parallel time", Proc. Sixteenth ACM Symp. on Theory of Computing, 1984, 249-257.

[Ajtai, Komlos and Szemeredi] M. Ajtai, J. Komlós, and E. Szemerédi, "An O(n log n) sorting network," Proc. Fifteenth ACM Symp. on Theory of Computing (1983), 1-9.

[Eckstein] D.M. Eckstein, "Simultaneous memory access," Technical Report TR-79-6, Computer Science Department, Iowa State University, Ames, Iowa, 1979.

[Even] S. Even, Graph Algorithms, Computer Science Press, 1979.

[Reif and Valiant] J. Reif and L.J. Valiant, "A logarithmic time sort for linear size networks", Proc. Fifteenth ACM Symposium on Theory of computing, 1983, pp. 10-16.

[Shiloach and Vishkin] Y. Shiloach and U. Vishkin, "An O(log n) parallel connectivity algorithm," J. Algorithms 3 (1982), 57-63.

[Tarjan and Vishkin] R.E. Tarjan and U. Vishkin, "An efficient parallel biconnectivity algorithm", Technical Report #69, Computer Science Department, New York University, New York, New York, 1983.

[Thompson] C.D. Thompson, "The VLSI complexity of sorting", IEEE Trans. Comput., (December 1983).

[Vishkin83a] U. Vishkin, "Implementation of simultaneous memory address access in models that forbit it," J. Algorithms 4 (1983), 45-50.

[Vishkin83b] U. Vishkin, "Synchronous parallel computation - a survey", Technical Report #71, Computer Science Department, New York University, New York, New York, 1983.

[Wyllie] J.C. Wyllie, "The complexity of parallel computation", Technical Report TR 79-387, Department of Computer Science, Cornell University, Ithaca, New York, 1979.

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142

PRINTED IN U S A

NYU/CompSci/TR-134 c.1
Atallah
Finding Euler tours in
parallel.

NYU/CompSci/TR-134 c.1
Atallah
Finding Euler tours in
parallel.

DATE DUE	BORROWER'S NAME

LIBRARY
N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012